

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**Detection of Code-Free Files**

**Inventors:**

Mihai Costea

Michael Sheldon

Zeke Odins-Lucas

and

Marc Seinfeld

ATTORNEY'S DOCKET NO. MS1-1903US

please remove paragraph 1 from the specification and renumber remaining paragraphs accordingly.

## RELATED APPLICATIONS

[0001] This patent application is related to U.S. patent application serial no. \_\_\_\_\_, titled “\_\_\_\_\_” filed on / / \_\_\_\_\_, commonly assigned herewith, and hereby incorporated by reference.

## TECHNICAL FIELD

[0002] This disclosure relates to detection of executable code-free computer files. [600]

## BACKGROUND

[0003] Complex computer file formats—which allow for extensibility and enhanced functionality—are becoming increasingly popular. Unfortunately, they also provide a vehicle within which authors of malicious viral software may hide malevolent executable code. To combat this situation, an “arms race” exists, wherein anti-viral (AV) software makers isolate copies of each new virus and obtain a “signature” for the new virus, so that it may be subsequently recognized.

[0004] Accordingly, anti-viral (AV) software is configured to scan input files looking for signatures of each known virus. Where no known signature is found, an input file is assumed to be clear of viral infection.

[0005] Unfortunately, it is frequently the case that a new virus will pass through the AV software because the AV software has not yet been updated to include the new virus. While the AV software makers tend to respond quickly, in many cases damage is done before they are able to respond with an upgrade, and before the consumer installs the upgrade. Accordingly, a need still exists for

techniques that are better able to prevent a new software virus from infecting a computer system.

## **SUMMARY**

[0006] Detection of code-free files is described. According to one implementation, an input file is parsed to recognize a file format. Contents of the input file are checked according to the recognized file format, if available, in an effort to determine whether executable code might exist within the input file. A status is then sent in response to the checking.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

[0007] The same reference numerals are used throughout the drawings to reference like components and features.

[0008] Fig. 1 is a flow diagram that describes an exemplary implementation by which code-free files may be detected, including a method employed for use in recognizing file formats and detecting executable code.

[0009] Fig. 2 illustrates an exemplary environment, wherein a code detection module is configured to provide information on executable code contained within a file to an email program.

[0010] Fig. 3 illustrates a second exemplary environment, wherein a code detection module is configured to provide information on executable code contained within a file to an instant messaging program.

[0011] Fig. 4 illustrates a third exemplary environment, wherein a code detection module is configured to provide information on executable code contained within a file to an Internet browsing program.

[0012] Fig. 5 illustrates exemplary detail of structure contained within the code detection module of Figs. 2—4, including an extensible parser module.

[0013] Fig. 6 is a flow diagram that describes an exemplary method by which the extensible parser module seen in Fig. 5 is extended.

[0014] Fig. 7 is a flow diagram that describes an exemplary method to detect code free files.

[0015] Fig. 8 is an exemplary computer system wherein a code detection module may be implemented.

## **DETAILED DESCRIPTION**

### **Overview**

[0016] The following discussion is directed to techniques for detecting code-free files. Detection of code-free files is advantageous, in that such files pose a greatly reduced security risk for users of email, instant messaging, Internet browsing and other applications. Where a file is known to be code-free, the user enjoys a higher likelihood that malicious “virus” software is not present.

### **General Process**

[0017] Fig. 1 shows the general process of detecting code-free files. At block 102, an input file is parsed to enable recognition of a file format by which the input file is configured. File formats are conventions by which data may be organized for use and storage; a number of such file formats are well-known, and are associated with file name extensions, wherein a limited and non-exhaustive list includes: jpeg, pdf, doc (Word®), vsd (Visio®), etc. At block 104, contents of the input file are checked according to the recognized file format, if available, to find executable code within the input file. At block 106, a status is sent according to results of the checking for executable code. Note that herein the term “code” or “executable code” is to be interpreted broadly and without limitation to examples cited, which include: processor executable instructions, scripts and other high-level languages, extensibility mechanisms and any other logic, device or mechanism which could be designed, corrupted or in any other way implemented to formulate a virus, worm or any other form of malicious, unauthorized, unwanted or unintended malware. In some implementations, the status (e.g. reflecting a file-has-code status, a file-has-no-code status or a don’t-know-if-file-has-code status)

may be sent to email, instant messaging, Internet browsing and other applications wherein the security from virus-infected software is advantageous.

### **Exemplary Environment**

[0018] Figs. 2—4 illustrate exemplary environments 200—400 within which a system to detect code-free files may be operated. In particular, Figs. 2—4 illustrate environments 200—400 within which an email client application 202, an instant messaging application 302 or Internet browsing program 402, respectively, are configured to receive information which may include one or more attached files 204. Note that the applications 202, 302, 402 are representative of a wide variety of hardware or software devices which could be configured to receive information from a code detection module 206. Additional representative devices include a firewall (hardware and/or software), a host intrusion detector (for use in a server, client, workstation, etc.), a host vulnerability assessor (for use in a server, client, workstation, etc.), a software backup management program, a CD and/or DVD burning program, a P2P (peer to peer) file-sharing program, or a variety of other applications. A code detection module 206 is configured to analyze the attached file 204 to determine if executable code is present. Depending upon the analysis, output of the code detection module 206 provides the application 202, 302, 402 with one of three possible inputs: a file-has-code status 208, a file-has-no-code status 210 or a don't-know (if the file has code) status 212. In the first case, the file-has-code status 208 reflects very probable recognition of a file format of the input file and discovery of executable code within the input file. Due to the dangers inherent with having executable code within a file, the application 202, 302, 402 etc. may use knowledge of this status to perform in a manner consistent

with these dangers. In the second case, the file-has-no-code status 210 reflects certain recognition of a file format of the input file and discovery of no executable code within the input file. Where the application 202, 302, 402 is assured that the file 204 is code-free, the user does not have to be troubled by dialog boxes or other aspects of a user interface requesting the user to decide if the file is to be trusted. In the third case, the don't-know (if the file has executable code) status 212 reflects failure to recognize a file format of the input file, and the resultant uncertainty of whether executable code exists within the input file.

### **Exemplary System**

[0019] Fig. 5 shows exemplary detail of the code detection module 206 seen in Figs. 2—4. The exemplary code detection module 206 may be configured in software, firmware or hardware, such as by an ASIC (application specific integrated circuit). An extensible parser module 502 may be formulated as a table configured to include a plurality of component parser modules 506(1)—506(N). The extensible parser module may be extended, such as by an exemplary method 600 seen in Fig. 6. Extensibility is desirable, since it is frequently the case that new file formats become known, or that interest in known file formats is increased. Accordingly, the extensible parser module 502 may be expanded to include an additional component parser 506(N+1) configured to recognize an additional file format and also configured to check for executable code within the new file format. Fig. 6 shows an exemplary process 600 by which the extensible parser module 502 of the code detection module 206 may be extended. At block 602, a file format is identified for addition to the extensible parser module 502. For example, it may be desired that the extensible parser module 502 be extended for

use with jpeg files. At block 604, a new component parser is configured according to the new file format (e.g. jpeg), wherein the new component parser is configured to recognize files of the new format and recognize executable code within such files. At block 606, functionality of the extensible parser module 502 is extended by addition of the new component parser 506(N+1) to an extensible table within the extensible parser 502.

[0020] Referring again to Fig. 5, the extensible parser module 502 is configured to contain a plurality of component parser modules 506(1)—506(N), wherein only two component parser modules are shown for reasons of illustrative simplicity. Each of the component parser modules 506(1)—506(N) is configured to recognize a file of a particular file format, and when the particular file format is recognized, is additionally configured to recognize executable code contained within the file. For example, parser module 506(1) may be configured to recognize a file format (e.g. a format by which data is organized for storage) of a Word® document. Parser module 506(1) may also be configured, upon recognition of the file format, to recognize executable code within an input file having a Word® file format. In this case, recognition of the file format assists in the recognition of the executable code.

[0021] Each of the component parser modules 506 includes a format investigation module 508, which is configured to parse the input file 204 and determine if the input file matches the file format for which the parser was configured to identify. The component parser modules also include a code section detector 510, which is configured to detect executable code within the input file particularly where that file is found to be of the file format associated with the component parser module.

[0022] Each component parser module 506 may also be configured to include three outputs, which indicate that the input file 204 has code 512, the input file has no code 514 and that it isn't known if the input file had code 516. In the implementation of Fig. 5, when the format investigation module 508 fails to detect a format associated with the input file 204, the component parser returns a don't-know status 516. When the file format was recognized, the output of the code detector 510 is used to determine if the input file has code 512 (i.e. code detector 510 found code) or the input file has no code 514 outputs is appropriate (i.e. code detector 510 did not find code).

[0023] The extensible parser module 502 also contains a controller or dispatch process 504, which is typically configured to: serve the input file to all available component parsers 506(1)—506(N); process the outputs of all of the component parsers; and send an overall response (i.e. code/no-code/don't-know) to an appropriate application. The controller 504 is configured to include a compound code section detector 518, which is configured to receive input from each component parser 506(1)—506(N) and to determine if any of the component parsers found code. Where code was detected by one of the component parsers 506(1)—506(N), output of the code detection module 206 will be the file-has-code status 208. A compound format investigation module 520 is configured to determine whether any of the component parsers 506(1)—506(N) recognized a format of the input file 204. Such an investigation is typically appropriate where none of the component parsers 506(1)—506(N) detected executable code. Where a file format was detected by one of the component parsers 506(1)—506(N), output of the code detection module 206 will be the file-has-no-code status 210.

Where a file format was not identified, don't-know (if the file has code) output of the code detection module 206 will be the don't-know status 212.

### **Exemplary Method**

[0024] An exemplary method 700 for implementing aspects of the detection of code-free files will now be described with primary reference to the flow diagram of Fig. 7. The method applies generally to the operation of exemplary components discussed above with respect to Figs. 2—4, and particularly Fig. 5. The elements of the described method may be performed by any appropriate means including, for example, hardware logic blocks on an ASIC or by the execution of processor-readable instructions defined on a processor-readable medium.

[0025] A "processor-readable medium," as used herein, can be any means that can contain, store, communicate, propagate, or transport instructions for use by, or execution by, a processor. A processor-readable medium can be, without limitation, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, device, or propagation medium. More specific examples of a processor-readable medium include, among others, an electrical connection having one or more wires, a portable computer diskette, a random access memory (RAM), a read-only memory (ROM), an erasable programmable-read-only memory (EPROM or Flash memory), an optical fiber, a rewritable compact disc (CD-RW), and a portable compact disc read-only memory (CDROM).

[0026] Fig. 7 shows an exemplary method 700 for detecting executable code within files. At block 702, an input file is parsed to enable recognition of a file format by which the input file is configured. The parser may be configured in a

compound manner, such as the compound or extensible parser 502 of Fig. 5, wherein the compound parser includes a plurality of component parsers 506(1)—506(N), each configured to recognize a specific file format.

[0027] At block 704, a determination is made if a file format has been recognized. If a file format has been recognized (following the Yes branch of block 704 to block 706), then at block 706 contents of the input file are checked according to the recognized file format to find executable code within the input file. Note that where the parser 502 is extensible and/or compound, the file format may match a file format for which one of the component parsers 506(1)—506(N) is configured to recognize. Accordingly, the controller 504 (Fig. 5) will evaluate information from each component parser 506(1)—506(N), when determining if a file format is recognized.

[0028] At block 708, a determination is made if executable code was found. If executable code was found (following the Yes branch of block 708 to block 712), then at block 712, a file-has-code status is sent, i.e. a file-has-code status is sent when the file format of the input file was recognized and executable code was found. The recognition may be made by any of the component parsers 506(1)—506(N). As seen by review of the structure of the exemplary code detection module 206 seen in Fig. 5, where the file format is recognized, a component parser is able to detect executable code, if present. Such code is inconsistent with file format, or is located according to the convention of the file format, and is therefore easily spotted. Accordingly, where one of the component parsers recognizes executable code, the controller responds by providing a file-has-code signal or message as appropriate.

[0029] At block 708, if a determination is made that indicates that no executable code was found (following the No branch of block 708 to block 710), then at block 710 a file-has-no-code status is sent when the file format of the input file was recognized and no executable code was found. Referring particularly to Fig. 5, it can be seen that if none of the component parsers recognized executable code within the input file, and the file format has been recognized by at least one of the component parsers, then the file-has-no-code status is registered. Note that a finding that the input file has been found to have no executable code is typically advantageous, since the absence of executable code assures the absence of malicious executable code, such as a virus.

[0030] Returning to block 704, if a file format has not been recognized (following the No branch of block 704 to block 714), then at block 714 a don't-know (if the input file has executable code) status 212 is sent when the file format is unknown. As seen by reference to block 212 of Fig. 5, where each of the component parsers was unable to determine the format of the file, the controller 504 is configured to issue a don't-know status to the appropriate receiver, such as an email application 202 (Fig. 2), an instant messaging application 302 (Fig. 3), an Internet browsing program 402 (Fig. 4) etc.

[0031] At block 716, in some applications, the component parsers 506(1)—506(N) may continue to parse the input file 204 even after one of the component parsers recognizes the format of the input file. This provides added security, in that, under rare circumstances, more than one component parser may make a valid recognition of a file format (i.e., a file could in rare instances be consistent with two different file formats). Thus in the rare circumstances wherein a second component parser recognizes the format of the input file, if either of the

component parsers recognizes executable code, the controller 504 can be configured to report that the input file-has-code. Alternatively, the compound parser can be configured to discontinue parsing when one of the component parsers recognizes the format of the input file. This tends to reduce time spent on the parsing operation.

[0032] As seen above, the file-has-no-code, file-has-code or don't-know status may be sent to email, instant messaging, Internet browsing and other applications wherein security from virus-infected software is advantageous. Figs. 2—4 illustrate exemplary uses for the code detection module 206. However, other uses for the code detection module are possible, such as in file storage applications wherein it is desired to store only executable code free files, etc.

[0033] While one or more methods have been disclosed by means of flow diagrams and text associated with the blocks of the flow diagrams, it is to be understood that the blocks do not necessarily have to be performed in the order in which they were presented, and that an alternative order may result in similar advantages. Furthermore, the methods are not exclusive and can be performed alone or in combination with one another.

### **Exemplary Computer**

[0034] Fig. 8 is an exemplary computer system wherein the exemplary code detection module and methods of operation of Figs. 1—7 may be implemented. Although one specific configuration is shown, the code detection module may be implemented in other computing configurations. The computing environment 800 includes a general-purpose computing system in the form of a computer 802. The components of computer 802 can include, but are not limited to, one or more

processors or processing units 804, a system memory 806, and a system bus 808 that couples various system components including the processor 804 to the system memory 806.

[0035] The system bus 808 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. An example of a system bus 808 would be a Peripheral Component Interconnects (PCI) bus, also known as a Mezzanine bus.

[0036] Computer 802 typically includes a variety of computer readable media. Such media can be any available media that is accessible by computer 802 and includes both volatile and non-volatile media, removable and non-removable media. The system memory 806 includes computer readable media in the form of volatile memory, such as random access memory (RAM) 810, and/or non-volatile memory, such as read only memory (ROM) 812. A basic input/output system (BIOS) 814, containing the basic routines that help to transfer information between elements within computer 802, such as during start-up, is stored in ROM 812. RAM 810 typically contains data and/or program modules that are immediately accessible to and/or presently operated on by the processing unit 804.

[0037] Computer 802 can also include other removable/non-removable, volatile/non-volatile computer storage media. By way of example, Fig. 8 illustrates a hard disk drive 816 for reading from and writing to a non-removable, non-volatile magnetic media (not shown), a magnetic disk drive 818 for reading from and writing to a removable, non-volatile magnetic disk 820 (e.g., a “floppy disk”), and an optical disk drive 822 for reading from and/or writing to a removable, non-volatile optical disk 824 such as a CD-ROM, DVD-ROM, or other

optical media. The hard disk drive 816, magnetic disk drive 818, and optical disk drive 822 are each connected to the system bus 808 by one or more data media interfaces 825. Alternatively, the hard disk drive 816, magnetic disk drive 818, and optical disk drive 822 can be connected to the system bus 808 by a SCSI interface (not shown).

[0038] The disk drives and their associated computer-readable media provide non-volatile storage of computer readable instructions, data structures, program modules, and other data for computer 802. Although the example illustrates a hard disk 816, a removable magnetic disk 820, and a removable optical disk 824, it is to be appreciated that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes or other magnetic storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or other optical storage, random access memories (RAM), read only memories (ROM), electrically erasable programmable read-only memory (EEPROM), and the like, can also be utilized to implement the exemplary computing system and environment.

[0039] Any number of program modules can be stored on the hard disk 816, magnetic disk 820, optical disk 824, ROM 812, and/or RAM 810, including by way of example, an operating system 826, one or more application programs 828, other program modules 830, and program data 832. Note that the code detection module 206 may be configured as an application program 828, a program module 830 or as a module located in another convenient location. Additionally, the input file 204 may be included among the data 832 or may be included in another convenient location. Each of such operating system 826, one or more application programs 828, other program modules 830, and program data 832 (or some

combination thereof) may include an embodiment of a caching scheme for user network access information.

[0040] Computer 802 can include a variety of computer/processor readable media identified as communication media. Communication media typically embodies computer readable instructions, data structures, program modules, or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term “modulated data signal” means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared, and other wireless media. Combinations of any of the above are also included within the scope of computer readable media.

[0041] A user can enter commands and information into computer system 802 via input devices such as a keyboard 834 and a pointing device 836 (e.g., a “mouse”). Other input devices 838 (not shown specifically) may include a microphone, joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and other input devices are connected to the processing unit 804 via input/output interfaces 840 that are coupled to the system bus 808, but may be connected by other interface and bus structures, such as a parallel port, game port, or a universal serial bus (USB).

[0042] A monitor 842 or other type of display device can also be connected to the system bus 808 via an interface, such as a video adapter 844. In addition to the monitor 842, other output peripheral devices can include components such as

speakers (not shown) and a printer 846 which can be connected to computer 802 via the input/output interfaces 840.

[0043] Computer 802 can operate in a networked environment using logical connections to one or more remote computers, such as a remote computing device 848. By way of example, the remote computing device 848 can be a personal computer, portable computer, a server, a router, a network computer, a peer device or other common network node, and the like. The remote computing device 848 is illustrated as a portable computer that can include many or all of the elements and features described herein relative to computer system 802.

[0044] Logical connections between computer 802 and the remote computer 848 are depicted as a local area network (LAN) 850 and a general wide area network (WAN) 852. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet. When implemented in a LAN networking environment, the computer 802 is connected to a local network 850 via a network interface or adapter 854. When implemented in a WAN networking environment, the computer 802 typically includes a modem 856 or other means for establishing communications over the wide network 852. The modem 856, which can be internal or external to computer 802, can be connected to the system bus 808 via the input/output interfaces 840 or other appropriate mechanisms. It is to be appreciated that the illustrated network connections are exemplary and that other means of establishing communication link(s) between the computers 802 and 848 can be employed.

[0045] In a networked environment, such as that illustrated with computing environment 800, program modules depicted relative to the computer 802, or portions thereof, may be stored in a remote memory storage device. By way of

example, remote application programs 858 reside on a memory device of remote computer 848. For purposes of illustration, application programs and other executable program components, such as the operating system, are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computer system 802, and are executed by the data processor(s) of the computer.

### Conclusion

[0046] Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.